MASTER THESIS

TITLE: Design of a Player-Plugin for Metadata Visualisation and Intelligent Navigation

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Alberto Fernández Chappotin

ADVISORS: Francesc Tarrés
Arnau Raventos
Raul Quijada

DATE: January, 28th 2019

Title: Design of a Player-Plugin for Metadata Visualisation and Intelligent Navigation

Author: Alberto Fernández Chappotin

Advisor: Francesc Tarrés
Arnau Raventos
Raúl Quijada

Date: January 28, 2019

Abstract

This project deals with the implementation of a metadata visualisation tool for the neural network developed by Ugiat Technologies. As it is a video processing software, the implemented tool is a player with extra features that allows to check if the metadata provided by the neural network is right.

It has been done using web technologies for better portability, mainly in JavaScript. This way almost any device can use it, but it is created for desktop use, no matter the operating system, since it runs in a web browser, but the screen resolution should be the one of a laptop or a desktop computer, no mobile usage is covered by the design.

For a better user experience, it allows voice commands to control some basic functions like playing, pausing and stopping the video as well as for enabling and disabling some of the extra features added to the video player. All the implementation is completely open source and it allows to continue adding functionalities.

The product obtained is good not just for helping the development and tests of the processing software, but also for making demos of its performance.

CONTENTS

INTRODUCTION

In the information era, people is consuming the traditional services in a new way. Almost all the interaction with the society is now taken through Internet. The capabilities of the global network are changing the way the people behaves. The access to any information is now matter of seconds. As shown in Fig. 1.1, the percentage of houses with an Internet connection in Spain has raised from 57.8% in 2010 to 86.4% in 2018 [1] that means a growth of around 5 million houses in 8 years [2].

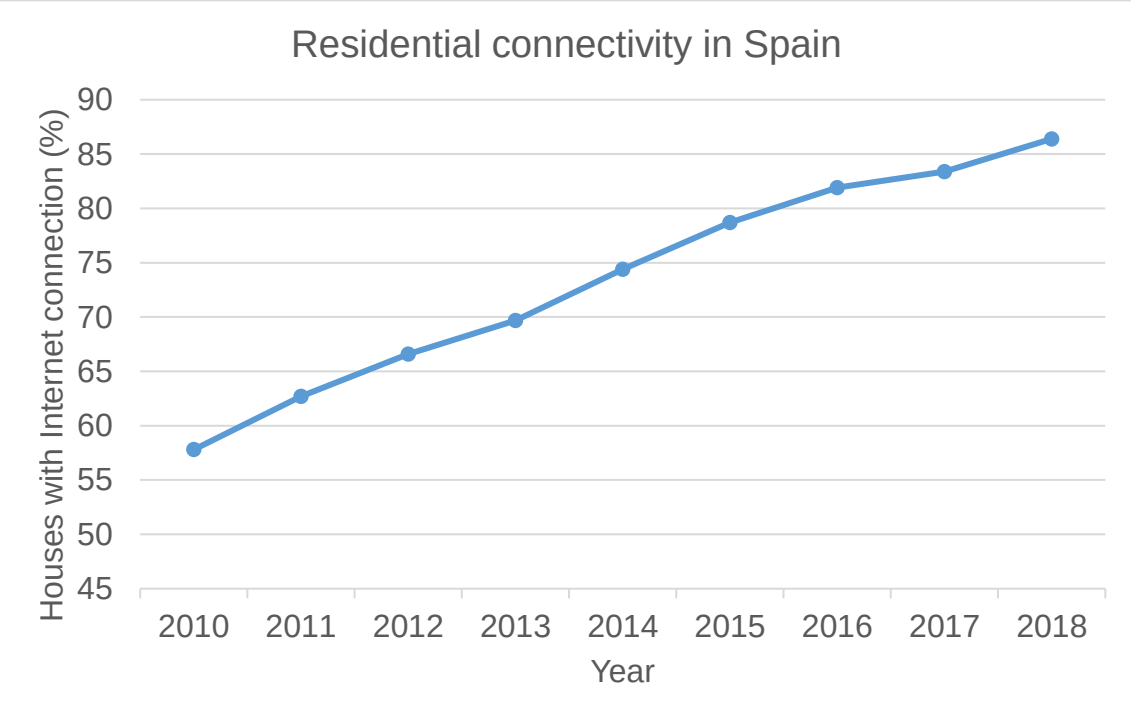


Fig. 1.1 Percent of homes with Internet connection in Spain.

This phenomena allows people to search for content not just in the traditional channels like TV or radio but also online, and when looking for information or news, Internet is preferred than the former channels. That's why most of the media is also showing their content in Internet.

By showing audio-visual content on the Internet, it is also possible to offer more information as added value, which cannot be implemented in other platforms. The fact that the content that is being shown is generated and visualized in computers with high computing capacity, allows the user experience to be improved using computational tools such as Artificial Intelligence and Big Data techniques.

In the specific case of videos, which is what this project will tackle, it can be used image and audio processing techniques to offer additional information or improve and organize the one that is already present in the video.

Ugiat Technologies is a spin-off that emerged in 2014 at the UPC and is dedicated to offering software to create this added value that can be used together with video to improve the user experience.

The software that analyses the video generates metadata that can be used during playback. All the results of the software analysis are stored in this metadata. The most relevant data extracted from a video are: faces (detection and recognition), objects present in the scene, place where the scene takes place, start and end of a scene, start and end of a camera shot; and many others that can be added.

The problem is that these metadata are generated in a format that is not human-friendly. Since the objective of the metadata is to be used in a machine-to-machine communication (between the processing software and the final application that will make use of the metadata), the format of the metadata is a typical format of minified serialised data, such as JSON, XML, CSV, etc. And this is very good for the performance of the final result, since a standard format is used for the exchange of information between both software stages. But in the development phase it can make the work of whoever is implementing it very complicated. A human will not be able to see at a glance if the metadata that is being generated is correct, or if it is synchronised.

This is why there is a need to have a metadata visualization tool, which allows to see the result that the final application would have if it is used the generated metadata. In this way, the developer will have a user-friendly feedback that will allow him to make the necessary adjustments in a more agile way.

Covering this need is the main objective of this project. Have a player that allows to visualize not only the video but also the metadata. In this way the developer will have a visualisation tool with which can check the correct functioning of the video processing software.

Being a tool for viewing metadata extracted from a video, it must evidently be an "improved player" that incorporates the elements of a traditional player, adding also the necessary parts for the analysis of metadata. This work also aims to make the player as easy-to-use as possible, incorporating navigation and visualization functions that take advantage of the loaded metadata. So that this player allows to be used not just as a tool during the development, but also as a demo of the developed technology.

CHAPTER 1. PROCESSING SOFTWARE

1.1. Machine learning

Artificial Intelligence (AI) and Machine Learning (ML) are two very common words nowadays. Almost every aspect of life is becoming involved by these technologies. With leading companies such as *NVIDIA*, *Intel*, *Google* or *Microsoft* at the forefront of this movement, the willingness to invest and develop in this area is becoming increasingly evident.

It isn't casual that these technologies are living their golden age. The era of Big Data has its moment now because for the first time there are enough computing capacity and a sufficiently large amount of stored data to exploit its potential.

Machine Learning is based in the idea of a software that is able to learn through training. The learning capability of can be defined as the generalisation of knowledge through experience. After the training, if everything went right, the system is able to provide the right output for new inputs.

Unlike other AI applications, ML isn't as visible to the end user, it is a core technology used to analyse information to help in understanding existing patterns and plan for future actions. Because ML is hidden from the end users, it creates a lot of opportunity for startups to try different models and techniques. Those companies can then partner with larger companies to leverage themselves into the business environment. While ML companies who market to specific niches have a chance to grow independently, the odds are that the ones who make the most inroads will be acquired by larger firms who realize that buying can be a more rapid entrance than building.

While the potential for ML is large, adoption is slow for the key reason of access to personnel. Machine learning is still very much a coding specialty, requiring frameworks such as *TensorFlow*, *Caffe*, and *Spark MLlib*, all of which need strong knowledge. The restricted hiring pool slow the ability for the knowledge to spread. In this scenario, *Ugiat Technologies* emerges as a company with a highly qualified team, capable of creating complex systems based on ML to offer them as a product or as a service to companies in other sectors.

The development carried out by *Ugiat Technologies* in the area of artificial intelligence applied to video processing, consists of a system composed of neural networks that perform a sequential processing of the video and offer high-level information regarding its content.

This chapter discusses the *Ugiat Technologies* product "smArDS", for which this project is intended. It will be explained the process through which the metadata of the video is obtained and the main market output and potential customers of this technology.

1.2. Basic concepts

Throughout this document will be mentioned concepts related to videos, which will be useful to comment initially to ensure a better understanding. Words like “scene”, “shot” and “keyframe” will be used to explain the structure of the metadata offered by the video processing software. These concepts have a relevant importance for the metadata processing, so, initially, the meaning of each one will be explained.

1.2.1. Scene

A scene is a temporal division of a video, in the case of films it is a unit of the narration, is an active entity happening on a stage. A scene changes when there is a discontinuity in time or when the location changes. In practice, a scene is a continuous part of a movie with:

- The same location
- The same actors
- The same continuity in the situation

Fig. 1.2 Frames of a scene.

In **¡Error! No se encuentra el origen de la referencia.** four frames of a single scene are presented. As it can be seen, the events are happening in the same room, with the same actors and without time jumps. That’s an example of a scene.

1.2.2. Shot

A shot is referred to a continuous camera recording without cuts or edits. In this way, a scene can have one or more shots. The shot can be considered as the

smaller temporal unit of a video, considering “video” as the transition of frames at a rate higher than 24 frames per second to create the illusion of movement.

Fig. 1.3 Frames of a shot.

¡Error! No se encuentra el origen de la referencia. presents frames of a single shot. It is a dialogue between two actors without cuts or edits. In this case the angle of the camera doesn’t changes. Even if it does, while there is no cuts, it still is a single shot.

1.2.3. Keyframe

A frame is a still image of a video, a sequence of frames in a highly enough rate is the phenomena that creates the video itself. A video can always be decomposed in all the frames that are part of it. The video processing software analyses the video by analysing its frames.

But commonly it isn’t necessary to analyse every single frame, mostly because in a shot, adjacent frames doesn’t differ too much. That’s why is usual to select some frames in shots to analyse and perform the video processing by the analysis of these frames.

Those “selected” frames are the ones called keyframes. For instance, any of the previous shown frames in **¡Error! No se encuentra el origen de la referencia.** and **¡Error! No se encuentra el origen de la referencia.** could be a key frame if it is selected to be processed by the software.

1.3. smArDS

smArDS is a product developed by *Ugiat Technologies* that puts AI at the service of advertisement. This system uses neural networks to automatically analyse video. This technology solves three main problems of videos for the Internet:

- Advertisement insertion
- Content-based navigation
- Thumbnail selection

The goal of smArDS is to provide a tool to provide advertisement that respects the viewer at the same time that improves brand quality and boosts the revenues of the ads.

Since smArDS is an implementation stage of a continuously growing software, in this project the processing of the video will be said to be done by the “processing software”.

1.3.1. Advertisement insertion

This is probably the main contribution of smArDS, due to the value that provides to the client that uses it. It is based in finding the right points to insert ads in videos. Very often in internet the ads are inserted randomly. Interrupting the action of the video in moments that the user is highly engaged with the content of it. This result very annoying for the viewer, and it has a negative effect for the advertiser, because the conversion rate of this kind of advertisement is very low.

What smArDS does is to obtain points in the video that represents changes in scenes and are good moments to insert ads, as shown in Fig. 1.4. Since the viewer is not very compromised with the video at that position he/she is more likely to watch the entire ad and maybe even clicking on it.

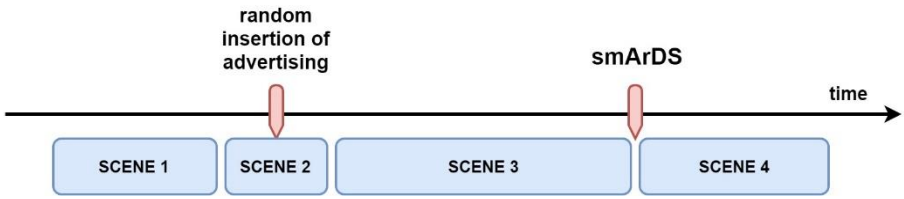


Fig. 1.4 Selection of points for ads insertion

Besides that, smArDS provides tools for contextualisation of advertisement. In order to present ads of products that are directly or indirectly related with the content of the video. With this strategy, viewer will be more suggested on those products and the overall conversion rate will increase.

1.3.2. Content-based navigation

When watching a video, viewer sometimes wants to see again a scene, or to skip the current one. Most players doesn’t provide a tool for that. Normally there will be a seek bar, for being able to go to any point and buttons to move a fixed amount of time backward or forward. This result in a messy navigation trying to find the starting of that scene that the viewer want to see.

By knowing where scenes starts and ends, navigation based on it can be implemented. This allows viewer to use a more powerful navigation that *Ugiat* calls smart navigation.

1.3.3. Thumbnail selection

Choosing a good thumbnail to a video in a video platform can define if the user will watch it or not. Sometimes this thumbnail is selected randomly, and it can be chosen a frame that is not the best for this purpose.

The solution proposed by smArDS is to select frames that contain big faces, with open eyes and preferably smiling.

1.4. Video processing

Several components are involved in the video processing, and can be separated into processes with well-defined functions. The sequential execution of these processes is what makes up the video processing. The following sections discuss each of these stages.

1.4.1. Getting shots of the video

The first step of the video processing is to obtain the shots of the video. In this stage the entire video is analysed by a convolutional neural network of three dimensions: time, width and height. The result is illustrated by Fig. 1.5 where it can be seen that shots can be, and often are, of different lengths.

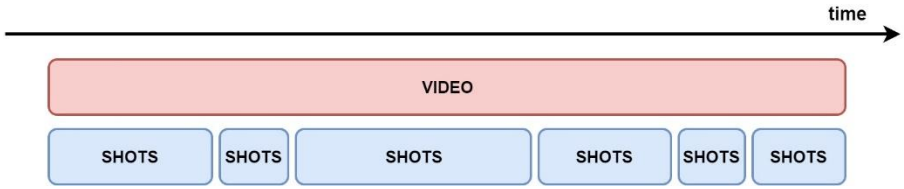


Fig. 1.5 Shot decomposition of a video

Since the shot is the most basic video unit (read section 1.2.2) this will be the basis of all the subsequent analysis.

1.4.2. Getting keyframes from shots

From each shot, the frames to be analysed will be selected. The strategy is to select at least three frames. Those will be: the first frame of the shot, the last frame of the shot and at least one frame between them. The result can be as presented by Fig. 1.6.

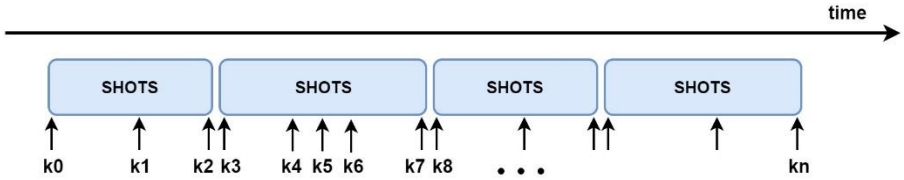


Fig. 1.6 Keyframe selection in shots of video

The criteria for selecting the middle keyframe(s) is by attending to movement vectors in the frames of the shot. If this shot has a high entropy due to high movement vectors, the system will select more than one keyframe. If the shot has low movement vectors just one keyframe will be selected, and it will be one considered as good in terms of its low movement vectors.

It is important to remark that for functionality of smArDS, just with this amount of keyframes is enough to provide the required information. But for demonstration purposes is better to select a uniform distribution of keyframes over the video. Common keyframe-to-frame rate is 1/10. This means that there is a keyframe every 10 frames. Considering a FPS of 29.97 which is a standard value, this is almost three keyframes per second. The result is an impression of continuity that is desired for live demos.

1.4.3. Getting scenes

To determine where there are the scene changes, the algorithm is to stand in a point of shot changing. Then take seven shots to the left and seven shots to the right as shown in Fig. 1.7.

To be considered as a scene changing point, three premises needs to be filled:

a) All the seven shots on the left are similar between them.
b) All the seven shots on the right are similar between them.
c) Shots on the left are different than shots from the right.

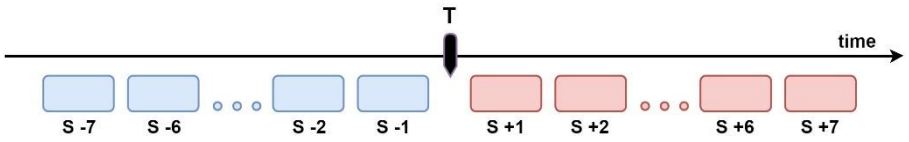


Fig. 1.7 Scene detection strategy

After this procedure, the points for advertisement are found. Those are precisely the points where scenes changes. This is the most important information for advertisers that use this technology.

1.5. Image processing

For being able to extract high level information from the video, it has to be analysed by frames, since image processing methods will be responsible for the detection of elements on video that provides context.

1.5.1. Objects and places

Objects and places are detected by using image classifiers over the keyframes. These image classifiers are performed by convolutional neural networks. The obtained objects and places of keyframes are very useful, because it is what allows to contextualise the ads that will be inserted. By means of this high level analysis, the system proposes certain type of advertisement based on the content the viewer is consuming.

1.5.2. Faces detection

Face detection, performed also by a neural network, consists in detecting the coordinates in video of the bounding boxes of human faces. This stage determines the presence or not of faces in an image (a frame) and if positive, provides the position of this face in the frame.

1.5.3. Faces recognition

Faces recognition goes one step further. It works over the previously detected faces and determines which faces belong to the same person. It labels the faces and reuse the label in case that a match is found. This face recognition is made by means of triplet neural networks.

CHAPTER 2. METADATA

2.1. Format

The metadata generated by the neural network that performs the video processing is obtained in minified XML format. In this way, a standard format for the exchange of metadata is guaranteed, in addition to optimizing disk space and bandwidth used for transmission.

2.1.1. XML

The Extensible Mark-up Language (XML) is a subset of the Standard Generalised Mark-up Language (SGML). Similar to HTML, this is a standard that has a W3C Recommendation [3].

The syntax of this format is quite easy to understand, especially because of its similarity to the popular HTML used on the web. An example of an XML file can be seen in Fig. 2.1, where it can be observed on the right side, the content of the document and on the left side, the hierarchical tree of the data in the document.

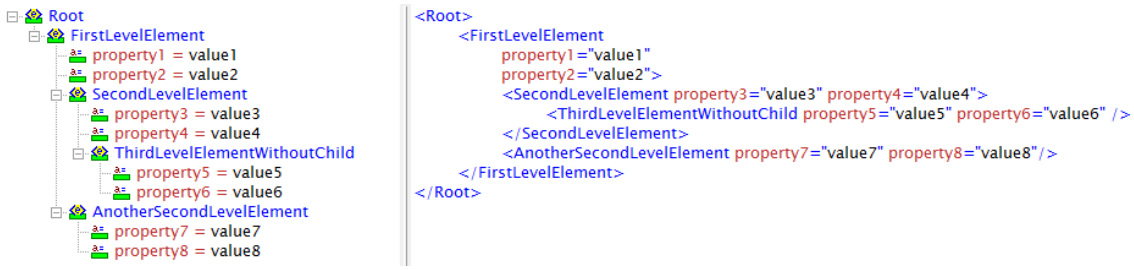


Fig. 2.1 Syntax of a XML document

As shown, this format allows to organise data in a structured way, making easier to access the required information at each moment. It is also an advantage of this format, the fact that the information is stored in a plain text file that makes it independent of hardware, software and platform.

2.1.2. Minified XML

The minification of data files is a technique increasingly recurrent in all applications that require information exchange. This procedure refers to the elimination of unnecessary bytes, such as additional spaces, line breaks and indents. By minimizing XML files, it is possible to speed up its download and analysis.

Of course, making these changes is detrimental to the human readability of the resulting text. It will continue to have the same hierarchical structure, but the ability to interpret at a glance by a human will be very affected. Fig. 2.2 shows the minified version of the same XML document presented in Fig. 2.1. Is easy to

check that the hierarchical structure is the same, however the plain text file now is harder to understand (by humans).

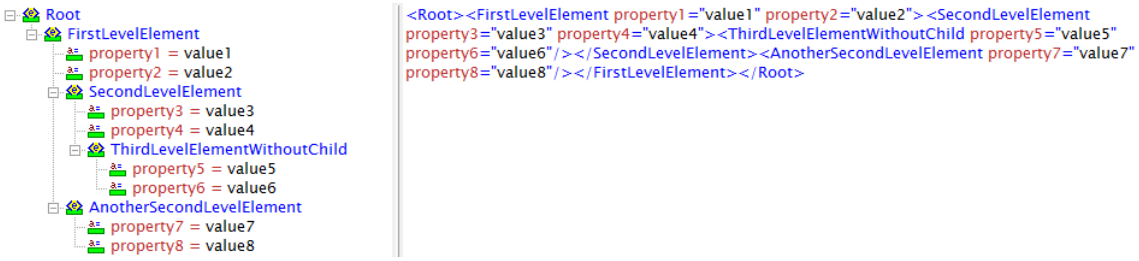


Fig. 2.2 Minified XML document

It is undeniable that minifying metadata provides many benefits for the performance of the final product. However, in files with a large amount of data, such as those generated by the video processing software, the resulting file does not provide comprehensible information on its own. In order to show the complexity of the metadata that is handled in this application, Fig. 2.3**¡Error! No se encuentra el origen de la referencia.** shows a small fragment of a file generated after the execution of the software.

<VideoDescription><GeneralInformation><SoftwareVersion value="1.03" /><VideoInformation><Info name="FileName" value="parlament_V2.mp4" /><Info name="FPS" value="25" /><Info name="Resolution" value="720x576" /></VideoInformation></GeneralInformation><TemporalGlobalDescriptors /><HierchicalDecomposition><SceneDecomposition><Scene category="scene" id="0001" tcin="00:00:00.01" tcout="01:04:03.05"><Shot category="shot" id="0004" tcin="00:00:43.11" tcout="00:01:02.12"><KeyFrameDecomposition><KeyFrame><Info name="Keyframe_ID" value="115" /><Info name="FileName" value="Shot 0004 - Keyframe 001086.jpg" /><Info name="FramePosition" value="1086" /><ImageDescriptors><FaceDescriptor><FaceInformation><Info name="NumberOfFaces" value="1" /></FaceInformation><Faces><Face face_embedding="[-0.14975588 -0.14902511 -0.01529277 0.0543507 0.02657004 -0.08576483
 0.02707412 0.01776832 -0.00283193 -0.04090789 -0.07920803 -0.02055672
 -0.00268909 -0.00287719 0.09901647 -0.0740195 0.01172777 0.11402581
 0.02361209 -0.03117171 0.12440928 -0.03730895 -0.10943859 -0.05591945
 0.07183599 -0.13271111 -0.14995778 0.02956093 0.13394196 0.02022338
 -0.02414685 0.14576149 -0.07448164 0.03964419 0.00919824 -0.02180927
 -0.04883448 -0.01105478 -0.0146877 -0.08953506 -0.24116711 0.07349726
 0.06949286 -0.0447846 0.1320126 0.00429467 -0.06683844 0.00499596
 -0.10895157 -0.02588253 0.1310707 0.0128521 0.11960822 0.08374999
 -0.10965703 0.0144098 -0.00814402 -0.06847268 -0.14320138 -0.01190599
 0.11530442 -0.12919952 0.02751225 0.15789348 0.13006626 -0.00490741
 0.02282041 -0.08512238 0.02154179 0.03397099 -0.12246436 -0.01142205
 -0.04147854 0.02154918 -0.15310499 0.07294432 0.04553046 -0.02190478
 -0.07024051 0.08481376 -0.00483656 -0.13928464 -0.0027346 -0.02253647
 -0.19897933 -0.00075652 0.14563441 -0.04740363 -0.07870457 0.10865126
 0.04122116 -0.24144058 0.04898896 0.05879852 0.01157082 0.05228977
 0.06163674 -0.03665437 0.06917033 0.0049077 -0.21649407 0.13764817
 -0.00747485 0.07601281 -0.15726933 0.05108231 0.32447523 0.05008347
 0.02716602 -0.07090552 0.03909086 0.08531129 0.01759192 0.02248057
 0.06428459 0.05876284 0.07228392 -0.06705449 0.08475098 0.07242584
 -0.00803593 0.07389417 0.07226901 -0.00139931 -0.02389211 0.02244516
 -0.02631257 0.00125398]" id="1" person_id="0" x1="0.335" x2="0.468" y1="0.442" y2="0.531" /></Faces></FaceDescriptor></ImageDescriptors></KeyFrame><KeyFrame><Info name="Keyframe_ID" value="116" /><Info name="FileName" value="Shot 0004 - Keyframe 001090.jpg" /><Info name="FramePosition" value="1090" /><ImageDescriptors><FaceDescriptor><FaceInformation><Info name="NumberOfFaces" value="1" /></FaceInformation><Faces><Face face_embedding="[0.00919932 -0.08365992 0.01644427 0.15971473 0.04451175 -0.19813618
 0.00294379 -0.03187659 -0.0308452 -0.05918599 -0.05989979 0.01423978
 0.05336056 -0.09012509 0.274999 -0.11684712 0.05351131 0.02999677
 0.04372676 0.02144299 0.17715637 -0.0204849 -0.13243173 -0.11875982
 0.00204841 -0.01529414 -0.09501037 -0.08192366 -0.04234817 0.03004318
 0.11528032 0.03681219 -0.01545775 0.02281918 0.06973746 0.05317165
 0.11236595 -0.10037542 -0.16973749 -0.13551226 -0.15810955 0.01958526
 0.03903893 -0.09307403 0.12213892 0.01441252 -0.13202465 0.00137351
 0.0193417 0.06119817 0.07410577 -0.01133068 0.05932648 0.14345573
 -0.05461291 0.17555735 0.09159758 -0.09042577 -0.04319514 0.00777347
 -0.00430058 -0.09527741 0.05160536 0.00779382 0.02389319 0.00032014
 0.13228038 -0.07796593 -0.08528118 -0.00694509 -0.14806409 -0.03775014
 -0.08339801 0.00151496 -0.03474288 -0.06364466 0.05469663 0.00759495
 -0.07939342 0.16576369 0.08315521 0.05682344 -0.07652255 -0.04425563
 -0.0343616 -0.08697043 0.00673098 -0.05462267 -0.02801147 0.04056411
 0.01362222 -0.25292668 -0.0054145 0.17372517 -0.04654136 0.03952905
 0.06836826 0.10879409 0.07356265 0.14738517 -0.10169269 0.04025967
 0.02118375 0.10582902 -0.05704485 -0.11982477 0.12076871 0.00051246
 0.02938188 -0.07500459 0.16507252 0.03707002 0.08764106 0.00962239
 0.10551891 -0.01478797 0.16452631 0.10694915 0.01535918 -0.03046356
 -0.05814609 -0.02923653 0.11219117 -0.02566355 -0.11426832 -0.05578811
 -0.06299102 -0.06303599]" id="1" person_id="0" x1="0.363" x2="0.490" y1="0.443" y2="0.529" /></Faces></FaceDescriptor></ImageDescriptors></KeyFrame><KeyFrame><Info name="Keyframe_ID" value="117" /><Info name="FileName" value="Shot 0004 - Keyframe 001100.jpg" /><Info name="FramePosition" value="1100" /><ImageDescriptors><FaceDescriptor><FaceInformation><Info name="NumberOfFaces" value="1" /></FaceInformation><Faces><Face face_embedding="[-0.03880138 0.01006374 -0.09077816 0.13191086 -0.0191061 -0.13308564
 0.03811673 0.03681705 -0.04573117 -0.0475874 -0.03144629 0.08059439
 0.14108099 -0.05063007 0.22096409 -0.12362818 0.01766989 0.00296433
 0.1164053 -0.00683476 0.15631488 0.06869857 -0.07835717 -0.05868678
 0.03371039 -0.00432694 -0.10885422 -0.04506505 0.1277435

Fig. 2.3 Fragment of metadata provided by the video processing

This example shows that the information that can be obtained by simply inspecting the metadata file is not useful to make decisions in an agile manner during the development stage. It is very complicated to check by a human if the results that are being shown are correct or not.

2.2. Structure

The structure of the metadata in the XML file is organised as follows:

```
<VideoDescription>
  <GeneralInformation>
    <SoftwareVersion value="_VERSION_"/>
    <VideoInformation>
      <Info name="FileName" value="_FILENAME_"/>
      <Info name="FPS" value="_FPS_"/>
      <Info name="Resolution" value="_RESOLUTION_"/>
    </VideoInformation>
  </GeneralInformation>
  <TemporalGlobalDescriptors/>
  <HierchicalDecomposition>
    <SceneDecomposition>
      <Scene category="scene" id="_ID_" tcin="_TCIN_" tcout="_TCOUT_">
        <Shot category="shot" id="_ID_" tcin="_TCIN_" tcout="_TCOUT_">
          <KeyFrameDecomposition>
            <KeyFrame>
              <Info name="Keyframe_ID" value="_ID_"/>
              <Info name="FileName" value="_FILENAME_"/>
              <Info name="FramePosition" value="_FRAMEPOSITION_"/>
              <ImageDescriptors>
                *****
              </ImageDescriptors>
            </KeyFrame>
            <KeyFrame></KeyFrame>
            <KeyFrame></KeyFrame>
            ...
          </KeyFrameDecomposition>
        </Shot>
        <Shot></Shot>
        <Shot></Shot>
        ...
      </Scene>
      <Scene></Scene>
      <Scene></Scene>
      ...
    </SceneDecomposition>
  </HierchicalDecomposition>
</VideoDescription>
```

Code 2.1 Main structure of metadata in XML format

Everything is contained in the *VideoDescription* mark that is like the root of the document. In the first level there are three labels:

- *GeneralInformation*
- *TemporalGlobalDescriptors*
- *HierarchicalDecomposition*

In Code 2.1 Main structure of metadata in XML format

, when the symbol "..." is used it refers to a repetition of the previous element at the same hierarchical level. For instance, the *KeyFrame* label, under *KeyFrameDecomposition*, appears for each key frame analysed. There is no information that let to know in advance how many key frames will be in a shot.

Moreover, inside the *ImageDescriptors* label appears the symbol "*****". It is used because the content of this label will be analysed in greater depth in the next section of this chapter and its structure will not be commented yet.

For a better understanding of the metadata, the structure shown in Code 2.1 will be top-down analysed in this chapter.

2.2.1. General information

This section of the metadata is under the *GeneralInformation* mark and its child nodes contain information about the processing software and the video processed. Specifically, the information currently presented in this section is:

- Video processing software version
- File name of the analysed video
- Frame rate of the analysed video (frames per second)
- Resolution of the analysed video (pixels in width and height)

2.2.2. Temporal global descriptors

This tag is not used yet. It is reserved to in a future, export global descriptors of the video. Currently, the processing software only analyses local descriptors. It means information related with a specific keyframe in the video.

However, the purpose is to improve the processing software to generate higher level descriptor that refers to concepts of the video as a whole.

2.2.3. Hierarchical decomposition

The *HierarchicalDecomposition* mark is the one in the metadata that contains all the results of the execution of the processing software. It is structured in temporal lapses that were recognised by the neural network.

The most complex temporal aggregation that the software recognises is a scene, that's why at the top level of the hierarchical decomposition there is a scene decomposition. Each scene has an id and the time in the video of its start and end.

The child nodes of a scene are shots. Each scene can have any quantity of shots. A shot also has a unique id and the beginning and ending time. With this hierarchical decomposition, the video is temporally divided as shown in Fig. 2.4, where can be seen that the video has three scenes of different duration, and each of them has different quantity of shots, even just one.

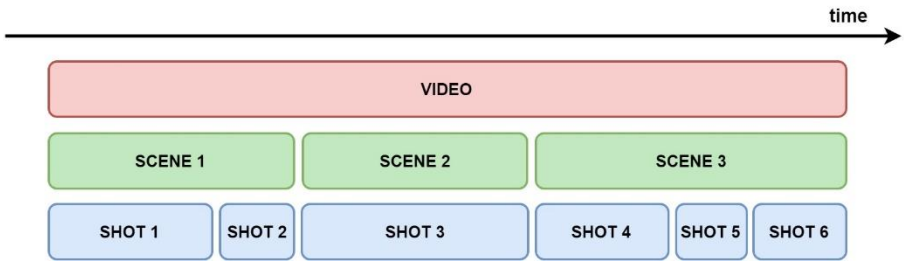


Fig. 2.4 Temporal hierarchical division of a video

The temporal unit smaller than shot is the frame, so the child nodes of a shot are the keyframes that were analysed in the video. During processing, each keyframe is stored as a picture, so besides the id, a keyframe also has as a property the filename of the corresponding picture. The value that contains the information about the position of the keyframe in the video is the *FramePosition* but it is not expressed in time but in number of ordered keyframes.

Of course, from this, the position in time can be obtained as the framerate is known. Dividing the frame position by the frame rate, the temporal position in seconds of the frame is obtained.

Inside the *KeyFrame* mark, there is a node named *ImageDescriptors* that is where the results of the high-level analysis are stored. For a better understanding, it will be approached in deep in the next section.

2.3. Image descriptors

The output of the video processing software, besides the delimitation in shots and scenes, is the detection of objects, places and faces in the video. The information about this last three elements is contained in their respective descriptors inside the keyframes where it has been detected.

2.3.1. Object descriptor

The neural network has a large list of objects that it is able to detect in an image. Each object is recognised with a certain confidence level. For each keyframe, the five objects with the highest confidence level are included as possible results.

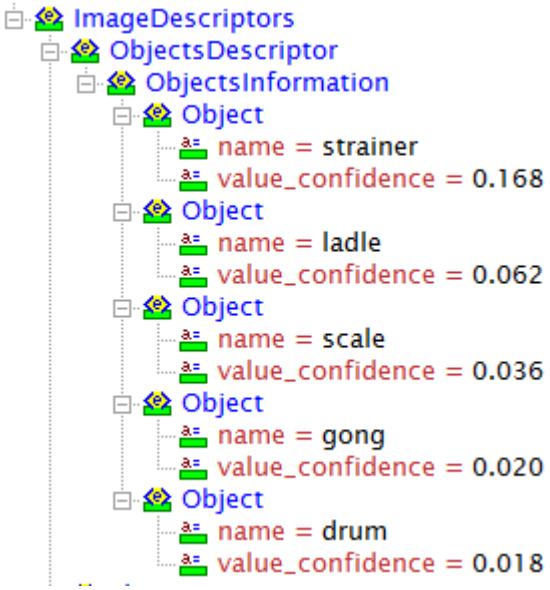


Fig. 2.5 Objects metadata structure

As it is shown in Fig. 2.5 the objects in the objects descriptor of the metadata are sorted in terms of confidence level. These objects have been detected in this specific keyframe, but depending on the video entropy, it can be applicable to a temporal vicinity of the keyframe.

2.3.2. Place descriptor

Unlike objects, places detected by the video processing software do not refer to elements that are in the image, but to the environment where the scene takes place. As well as in objects detection, the neural network has a lot of places that can be recognised and it returns the five with highest confidence level.

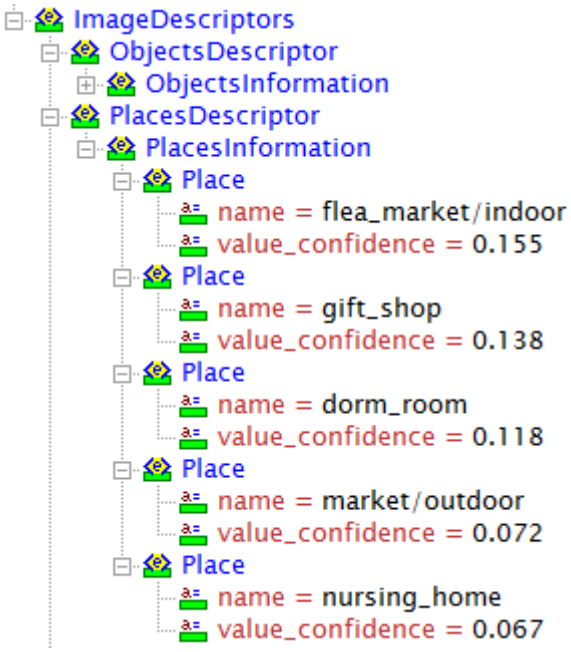


Fig. 2.6 Places metadata structure

As can be seen, Fig. 2.6 is very similar to Fig. 2.5**¡Error! No se encuentra el origen de la referencia.** in terms of structure and sorting.

2.3.3. Face descriptor

Faces behaves unlike objects and places. In a keyframe objects and places will always be found, but that's not the case of faces. Since a keyframe can contain many faces, or no one, there is no pre-established amount of faces to appear in metadata.

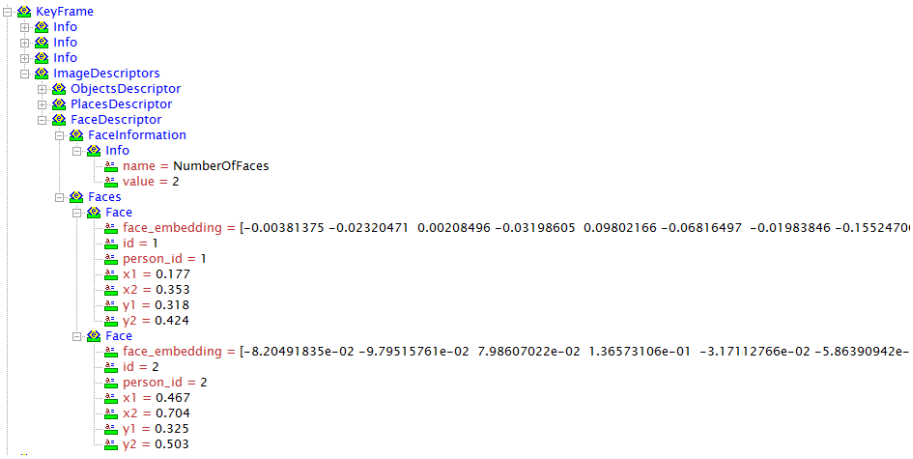


Fig. 2.7 Faces metadata structure

In the example shown by Fig. 2.7 there are two faces detected. For each of them there is information about the position in the image, given by the coordinates of the rectangle that contains the face (*x1*, *x2*, *y1* and *y2*).

The *id* parameter represents a unique identifier of the face in this keyframe. On the other hand, the parameter *person_id* represents the face recognition capability of the software. When a face is detected in the video, a label is attached to it, for simplicity the labels are consecutive natural numbers. If another face is detected, the software recognises if it is the same face or a new one and assigns it the same label or a new one respectively.

The *face_embedding* parameter is the result of the processing of a face. For each detected face, the processing software generates a vector of numbers. Similar faces produces similar numbers. The implementation of face recognition is made by means of clustering algorithms that may be implemented computing the Euclidean distance between vectors. If they are close enough it can be guessed that is the same person. That's how the video processing software recognises faces.

CHAPTER 3. CURRENT TOOLS

3.1. Platform

The metadata visualisation tool that this project aims to create should be as usable as possible. That means also that shouldn't be platform dependent.

Following this idea, the most reasonable option is to do it with web technology. That will allows to use it from any place and any device with a simple browser.

As it has to run on client side, the players that will be analysed are those has an API for interact with them through JavaScript.

3.1.1. Why JavaScript?

Like all computer languages, JavaScript has certain advantages and disadvantages. Many of the pros and cons are related to JavaScript executing often in a client's browser, but there are other ways to use JavaScript now that allow it to have the same benefits of server-side languages. This feature is making JavaScript more and more popular nowadays and even when the purpose of this project is to use it just in client-side, since there is some current resources already developed on Apache servers, it is good to use state of the art technologies.

JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server. It also has no need to be compiled on the client side which gives it certain speed advantages (granted, adding some risk dependent on that quality of the code developed). It is used everywhere in the web. The resources to learn JavaScript are numerous. StackOverflow and GitHub have many projects that are using JavaScript and the language as a whole has gained a lot of traction in the industry in recent years especially.

JavaScript plays nicely with other languages and can be used in a huge variety of applications. Unlike PHP or SSI scripts, JavaScript can be inserted into any web page regardless of the file extension and can also be used inside scripts written in other languages such as Perl and PHP.

3.1.2. Why PHP?

As well as JavaScript was chosen for client-side, the project will need some logic in server-side. This becomes clear when emerges the need to save files to storage dynamically. There are many options to develop the server-side software, in this case, PHP was selected to use.

PHP results in faster site loading speeds, its code executes quite fast since it runs in its own memory space. In working with this language, most tools associated with the program are open source software, so there is no need to pay for them. PHP would only require running on a Linux server or some apache server available for Windows as the one included in the XAMPP package. In any case it is free to install in any computer and available through a hosting provider at no additional cost. The community of PHP is very large and there is a lot of documentation available for free.

3.2. Available players

When HTML5 introduced the <*video*> and <*audio*> tags, they made media files genuinely accessible to the Internet. HTML5 videos are fast replacing Flash Player and other similar third-party media players. Historically, it's been quite a cumbersome process to get media to play correctly. Often the <*embed*> and <*object*> tags would need to be used which would assign an extensive list of parameters to get the media playback working.

For achieving the goal of creating this tool, the best starting point is a web player that allows to develop over it, to create plugins or extensions. There are some players with this feature, some of them are free and others not. In this section some of the players that were considered will be shown.

The basic controls that we need for the player are:

- Play and pause events.
- Play, pause and seek methods.
- Capability to get current position of the video.
- Capability to add buttons to player with custom functions.
- Capability to draw over the image in the video as an overlay.

The players that will be presented were considered taking into account if they were able to offer this features and if it is possible, the difficulty of implementing it in each one.

3.2.1. HTML5 Video player

Obviously, the first considered player was the simplest one that comes with the HTML5 standard. It is a simple and powerful player that would require no third party libraries.

The difference with HTML5 video tag is that it handle files as images. Attributes like *height*, *width* and *autoplay* can be defined in the tag similar to any other HTML element.

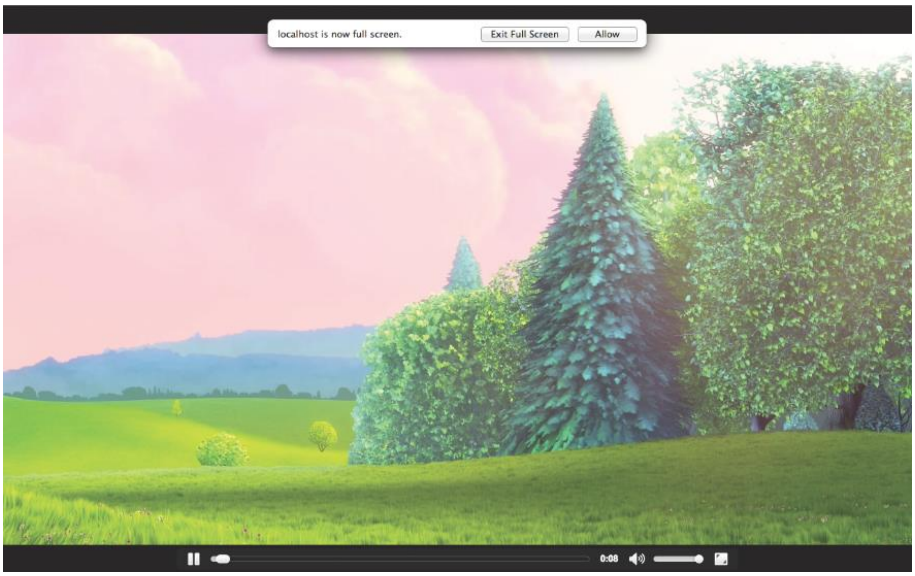


Fig. 3.1 HTML5 Video player

The main advantage of this player is that it comes with the HTML5 standard, so it will need no extra libraries just for the player itself. It has the needed events for handling the behaviour of the player and the utilization of the metadata.

The use of this player in a web page is as simple as shown in Code 3.1 where it is used in the simplest way possible.

```
<video src="url" width="640px" height="380px" autoplay/>
```

Code 3.1 Use of the HTML5 video tag

In HTML5, there are 3 supported video formats: MP4, WebM and Ogg. It is recommendable to have the video in at least two of them for guaranteeing that the video will be shown in any browser. The way to do it is as in Code 3.2 where the browser will try first with the MP4 and if it is not supported, will try with Ogg.

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

Code 3.2 Specify alternative video format if the first isn't supported by browser

In the case that no one of the provided video formats is supported by the browser, the text "Your browser does not support the video tag" will be shown in the position of this element.

Nevertheless, MP4 is the format that most of the browsers support. WebM and Ogg can be not supported by Internet Explorer and Safari. But they support MP4, and also does it Google Chrome, Mozilla Firefox and Opera (from version 25).

The main issue with the HTML5 video player is the lack of methods for controlling the video and the difficulty to add more control elements in the user interface. This was the main reason why it wasn't chosen for developing the tool

3.2.2. Video.js player

In the case of video.js it is a very powerful player with a quite good documentation on its API and completely controlled with JavaScript, both in server and client side.

In the documentation of video.js is divided in "Guides" and "API docs". Guides explain general topics and use cases (e.g. setup). API docs are automatically generated from the codebase and give specific details about functions, properties, and events.

The main advantages of this player, shown in Fig. 3.2, are the amount of events and versatility of the API for controlling design and behaviour.

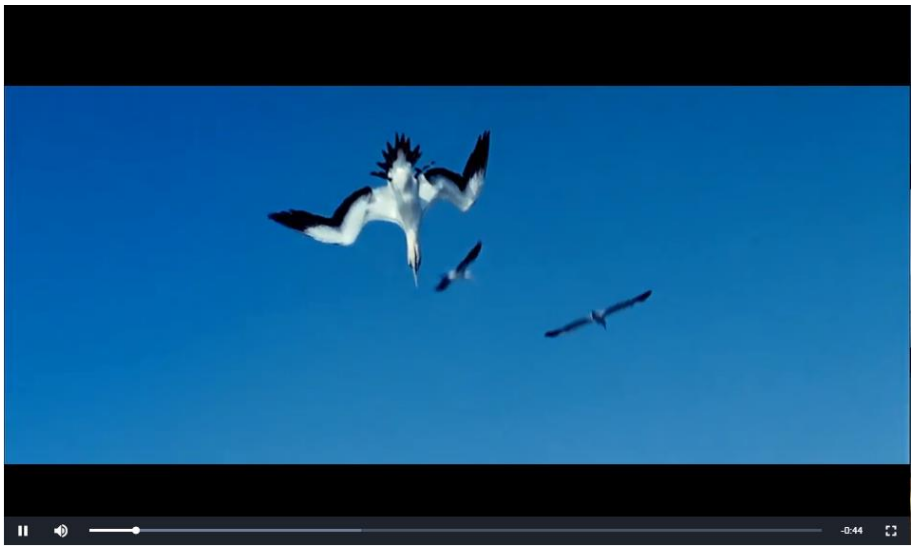


Fig. 3.2 Video.js video player

In any case, video.js is a very interesting player that tackles the lacks of the standard HTML5 player.

3.2.3. Plyr

Plyr is a customisable and straightforward HTML5, Vimeo and YouTube media player. It supports all modern browsers. The complete source can be accessed with NPM.

Fig. 3.3 Plyr video player

Plyr is very customisable and responsive. It even support monetisation and streaming. Its API allows toggle playback, volume, seeking, and more.

3.2.4. JW Player

JW Player is an extremely lightweight web video player that is completely implemented with JavaScript. Its creators boast that JW is the fastest player.

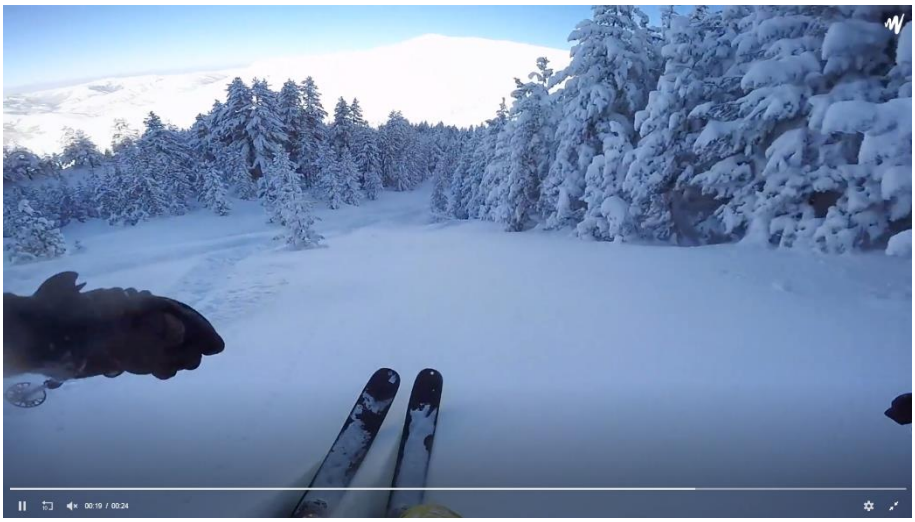


Fig. 3.4 JW Player

But the advantages of JW player aren't just about speed. It has a very powerful API and a lot of events that allow to develop with plenty freedom over it.

Also, the companies that are supporting this player offers the warranty of quality. Names like AWS, FOX, Univision and Business Insider make JW Player very interesting for this project goal.

In the newest version of JW player there isn't a free license. Starting at 30 USD per year with a limited amount of videos and plays. But there are older versions

available that are fully functional and have all the required functionalities for the metadata visualisation tool. The recent versions of JW Player are focused in security with self-implemented password protection sharing of videos. But that's not the scenario this project tries to cover, so it is not needed to use the latest version of the player.

Because of the capabilities and lightweight of JW Player, it was the chosen one to develop the metadata visualisation tool that this project aims to create.

CHAPTER 4. IMPLEMENTATION

4.1 Handling metadata

As the used language is JavaScript. The format of the metadata isn't the best one for parsing to an object in this language. Even when XML is a very common format, JavaScript has its own serialisation format: JSON.

In runtime, the software will not interact with the metadata files. The best thing to do is to load all this information in memory for a faster access to it. For simplicity, the metadata will be loaded into an object and all the functionalities of the software will address this object.

Of course, when the page loads in the browser, the metadata file has to be parsed into this object. And for that, is faster to do it from a JSON file than from a XML file. So, the flow chart of the execution when loading will be like the one presented in Fig. 4.1.

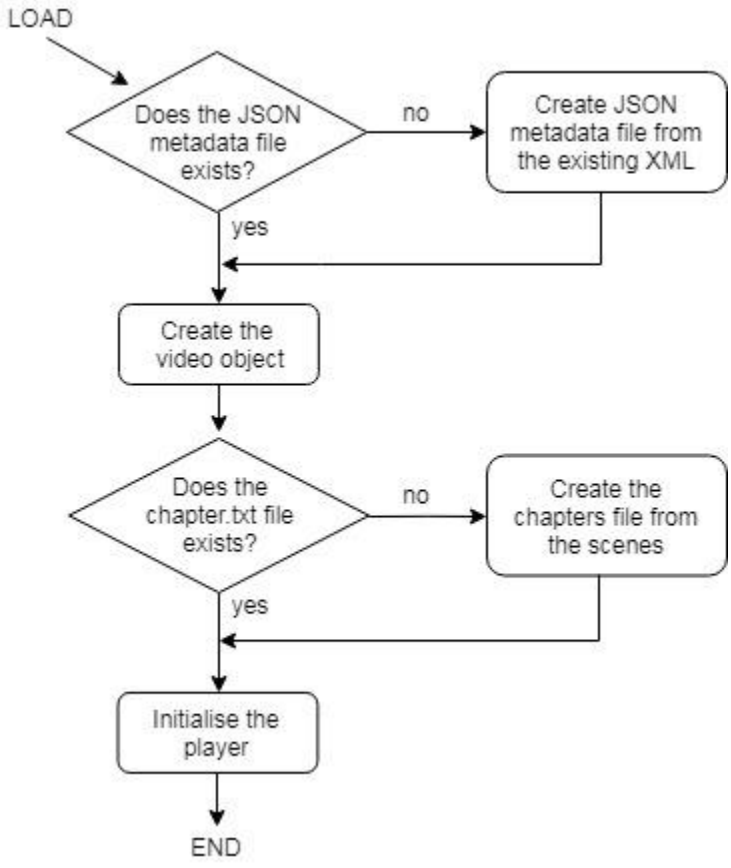


Fig. 4.1 Flowchart of player load

As shown in the previous figure, the program attempts to load a JSON first and creates the object from it. If it doesn't exists, meaning that this is the first time that the metadata is loaded, the program creates the JSON equivalent to that XML. This way, the first time the page is visited, the load takes a bit longer, but in the next times, the load is faster than if it were made always from the XML file.

A simple analysis of the performance of the page load in a given video and its metadata the load of the page took 8.42 seconds the first time it was visited. It means that it was the time that the generation of the JSON file and the chapters file took place.

The time distribution, according to the Google Chrome developer tools, was like the one presented in Fig. 4.2.

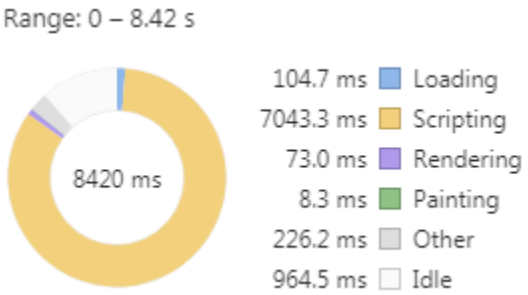


Fig. 4.2 Profiling summary of the first load of the webpage

In this execution, the function that performs all the logic commented before ran for a total time of 4456.51ms and specifically the function that creates the object from the XML file took 296.16ms.

It should be taken into account that in this scenario, the functions to check if files exists takes long time, because since it is a script running on the client and asking for the existence of files in the server, these checks are actually HTTP requests. Also, when files are generated, like the JSON file, to store it in the server a HTTP POST request with the content of the file is made.

The most delaying issue with this problem is that it cannot be done asynchronously, because the next step after the metadata load is the player initialisation and for that the files need to be accessible. So the main thread of the script have to wait for this file.

Once the webpage is fully loaded for the first time, the JSON file has been created, so it will never use again the XML file. Repeating the execution analysis when loading the webpage for the second time, the results are much better. As can be seen in Fig. 4.3 the total time has been reduced to less than the half.

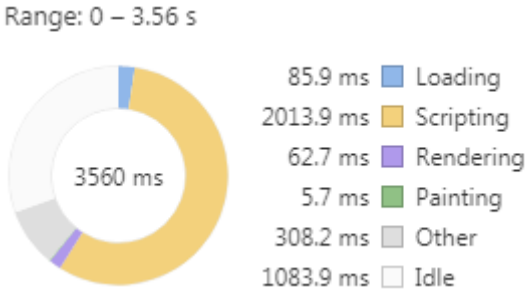
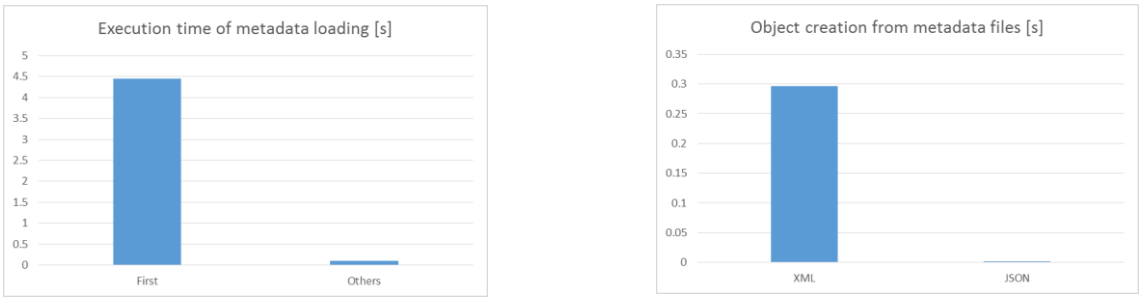


Fig. 4.3 Profiling summary of the second load of the webpage

In this case the execution of the function that performs all the logic took 101.11ms and specifically the creation of the object from the JSON file took 2.01ms.

It means that the total time for the logic is 2.27% of the first execution. But the improvement that really matters is that the creation of the object from a JSON file takes 0.68% of the time that it would last if using a XML file.

These values become clearer when seen graphically, in Fig. 4.4 (a) it can be compared the time that take the metadata load into memory for the first time the page is loaded compared with the following times. In Fig. 4.4 (b) the same time cost comparison is made, but this time comparing the time cost of the object creation when using a XML file against the time when using a JSON file.



(a) (b)

Fig. 4.4 Comparison of time cost for metadata loading

This difference is so notable because parsing JSON file into an object is native in JavaScript, while doing the same from XML files needs a custom parser that will always be less efficient.

At the end, considering that the penalty of this method is just in the first execution (see Fig. 4.4 a) and the benefits it provides are in every execution (see Fig. 4.4 b), the method of creating a JSON file equivalent to the XML provided by the video processing software is considered useful for the metadata visualization tool.

4.1.1 Reading files dynamically from storage

Reading files in JavaScript is unlike other languages. Since it is a script running in a browser, it has no access to the files in the user device and neither in the web server or at least not directly.

This means that for reading a file that is in the web server storage, or even just checking if the file exists, a HTTP request has to be made to ask the server for the file. This action can be done using a technique called AJAX.

With AJAX, JavaScript will make a request to the server, interpret the results, and take actions based on the server response. In the purest sense, the user would never know that anything was even transmitted to the server. This technology is independent of web server platform.

To read a file, the program uses an URL instead of a path in a file system. By sending a HTTP GET request to that URL, the content of the file will be obtained in plain text. If it is only needed to know if the file exists, just by reading the HTTP response code in the headers of the server response is possible to get the answer. An existing file URL will produce a HTTP 200 server response, while a non-existing file will result in a HTTP 404.

4.1.2 Saving files dynamically to storage

Writing files to web server is similar to reading in terms of the need of sending HTTP requests. But unlike reading, when writing there is more than a simple URL request to the server. The content that should be written in the file has to be sent.

For that purpose is better to use a server-side script that receives via POST the name of the file and its content. Through JavaScript POST requests can be sent asynchronously (AJAX) and this server-side script written in PHP will receive this data and save the file, since PHP is running on server and it does have direct access to files.

4.2 Video object structure

As said before, for the execution of the metadata visualisation tool, all the information has to be easy and rapidly accessible to allow to work with it in real time. The way to do that is by loading all the information in memory.

To do so, the best way is to create an object that contains all the information in a structure that makes it easy to access. The name of this object will be *video*, as it has the it has the information relative to it and the structure will be the one described in Code 4.1 Structure of video object as JSON string

where it can be seen how this object was designed to improve player performance.

It is important to remark that the structure is presented in JSON notation, so this is the “pretty” print of the JSON file that is created as commented in Section 4.1.

```
{
  "filename": <string>,
  "fps": <int>,
  "duration": <double>,
  "resolution": [<int>,<int>],
  "shots": [
              {
                "cathegory": <string>,
                "id": <int>,
                "position": <double>
```

```
                },
                ...
            ],
    "keyframes": [
                {
                  "id": <int>,
                  "position": <double>,
                  "facesQuant": <int>,
                  "objects": [
                                {
                                  "name": <string>,
                                  "confidence": <double>
                                },
                                ...
                              ],
                  "places": [
                                {
                                  "name": <string>,
                                  "confidence": <double>
                                },
                                ...
                              ],
                  "faces": [
                                {
                                  "id": <int>,
                                  "personId": <string>,
                                  "x1": <int>,
                                  "x2": <int>,
                                  "y1": <int>,
                                  "y2": <int>
                                },
                                ...
                            ]
                },
                ...
              ],
    "scenes": [<double>,<double>,...],
    "chapters": <string>
}
```

Code 4.1 Structure of video object as JSON string

The first attributes of the object are simple properties: *filename*, *fps*, *resolution*, *duration*. In each of the attributes of the object shown, the data type of the possible value has been placed in its position.

The first complex attribute is *shots*. It is an array of the shots of the video. Each shot contains its *cathegory*, *id* and *position*. This position and all the temporal positions of elements in the object, are expressed in seconds.

After *shots* the next attribute is *keyframes*. It is also an array, this time of the keyframes of the video. The keyframe contains the main information of the metadata. Every keyframe has *id*, *position*, *facesQuant* (referring to the quantity of faces detected in that keyframe) and three arrays: *objects*, *places* and *faces*.

Objects and places detected in the keyframe are presented in an identic structure, they are both arrays of a known length of five elements and each element contain

two properties: the *name* of the object or place and the *confidence* that this object or place was detected with. On the other hand, the *faces* array has a variable length, expressed in *facesQuant* and can be empty. In case of having elements, each detected face has an *id*, *personId* and the coordinates of this face in the screen expressed by *x1*, *x2*, *y1* and *y2*. The attribute *personId* is the one that serves to show face recognition. The coordinates determines the face position in the keyframe in the way shown in Fig. 4.5.

Fig. 4.5 Coordinates of a face detected in the keyframe

The values of these coordinates are expressed in percentage (divided by 100) of the respective dimension.

4.3 Displaying metadata in real time

As commented in section 2.3, the neural network that performs the video processing is able to recognise objects, places and faces. But it does it by analysing keyframes that are images. This means that there is no continuous information about the elements in the video, but just in the keyframes position. Of course, if all the frames of the video were analysed, all of them will be keyframes and there will be metadata information in every single frame of the video, but this is computationally expensive.

Instead of doing so and based on that keyframes are not too far (in time) from each other it can be considered that the objects places and faces of any position will probably be the same of the closest keyframe to that position. This is the assumption on which the metadata visualization tool will be based. In order to do so, one of the most recurrent and important procedures to do will be to find the closest keyframe to a given position.

4.3.1 Getting the closest keyframe

The simplest way to get the closest keyframe to a given position is by asking for the distance (in time) to each keyframe starting from the first and tracking the

behaviour of that distance. The distance will be decreasing while the closest keyframe is closer, and start increasing after passing it.

Fig. 4.6 shows how keyframes are distributed in time and how a given position in the video will always be closer to one keyframe.

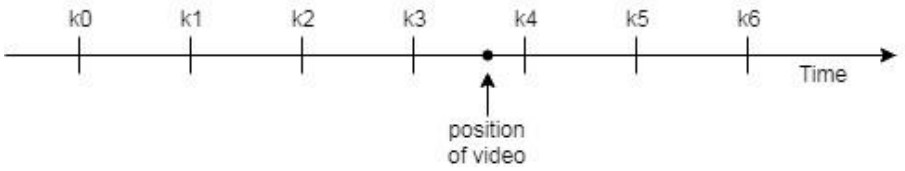


Fig. 4.6 Temporal distribution of keyframes in video

For instance, for getting the closest keyframe to the position of the video in Fig. 4.6 using the algorithm explained, the procedure is to obtain the distance in time to each keyframe. The distance in time is simply the absolute difference between the position of the video and the position of the keyframe.

Let Dk_N be the absolute difference between the position of the video and the nth keyframe. From Fig. 4.6 it can be checked that:

$Dk_0 > Dk_1 > Dk_2 > Dk_3 > Dk_4 < Dk_5 < Dk_6$

In the program, every time a new distance is obtained it is lower than the previous one. Just after obtain the distance to the closest keyframe, the following distances are higher. Then, the stop condition of this algorithm is that the new obtained distance is greater than the previous one. That means that the previous keyframe is the closest to the current position.

As said before, this is the simplest way, is the analogue to a linear search. But, as the linear search it is not the most efficient way to find the closest keyframe. The problem with this method is that the time that takes to find the closest keyframe depends on the position of the video. For positions close to the beginning of the video it will be really fast, but the closer to the end of the video, the longer it takes to find the keyframe.

To tackle this issue and taking advantage of the fact that the keyframes are chronologically sorted, an analogue to the bisection method can be used. The algorithm is to reduce the set of keyframes that contains the current video position. Fig. 4.7 helps to visualise the algorithm.

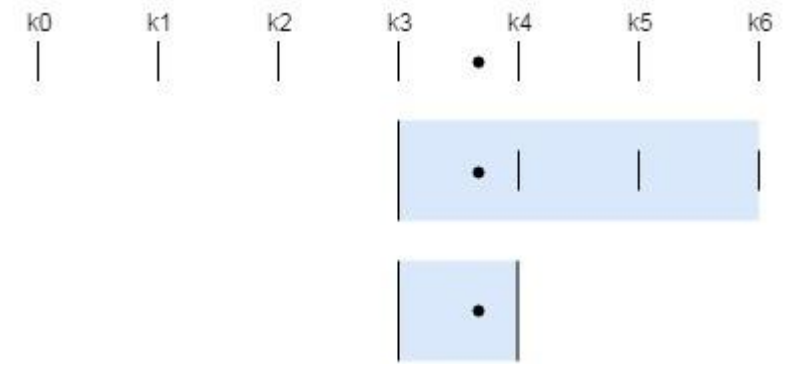


Fig. 4.7 Bisection method for finding the closest keyframe

In each iteration, the centre of the keyframes set is calculated. As it is a discrete variable, when the set has an even quantity of keyframes, the middle keyframe can be selected as integer part of the value obtained as the centre. Then, it is compared the position of the middle and the current position of the video, to determine if the current position is on the left or the right half of the set. Once the half if chosen, the process is repeated. In every iteration the set will be reduced to the half. The stop condition is when the set has only two keyframes, that means that the current video position is between those two keyframes and the distance to each one of them is computed. The one with the smaller distance will be the closest keyframe.

The benefits of using bisection instead of linear method is pretty clear, nevertheless, for showing how important is this improve, a simulation with Matlab was made. The test consists in computing the average quantity of iterations needed to find the closest keyframe to a given position in a video of N keyframes. In the case of the bisection method, the worst case was considered. It means that every time the set was divided in subsets of different sizes, the current video position was in the biggest. Even so, in Fig. 4.8 can be seen that the iterations needed for bisection method are always equal or lower than for linear method.

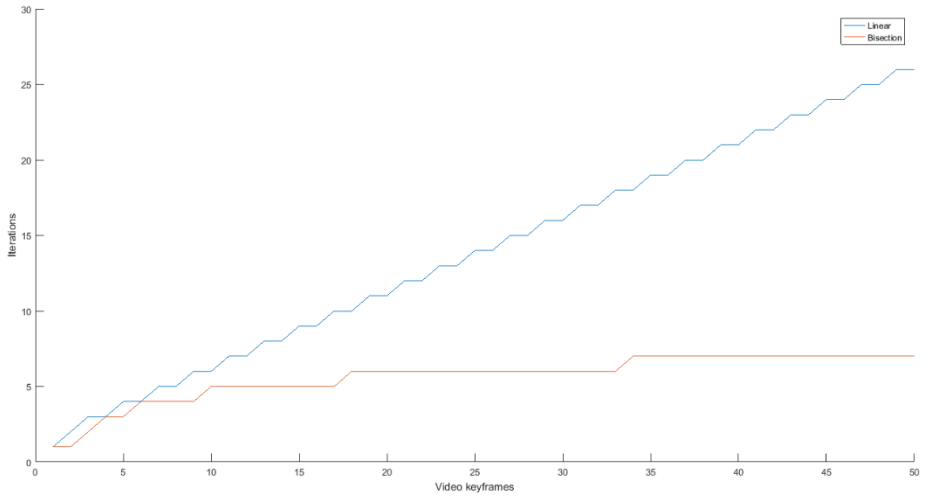


Fig. 4.8 Linear and Bisection methods for finding closest keyframe

It is also remarkable, that the quantity of keyframes analysed by the video processing software is much greater than the values in the graph, so the benefits of using bisection are even more noticeable than in the simulation.

Of course, it should be noticed that this function will just be needed when user jumps to a video position. While playing the problem is solved a lot easier. Since playing the video is advancing in just one direction, there is a simple way to get the closest keyframe: always save the current closest keyframe, and in each frame check if the next keyframe is already closer than the current one. This is so simple because there are only two candidates, the current keyframe and the next one. The formerly explained bisection method is used just when the user jumps in time the video, by meanings of step forward or backward buttons or by clicking on the desired position of the seek bar.

4.3.2 Scenes and shots

As said in Section 1.4, the processing software splits the video in scenes and shots. This feature is useful for smart seeking, since it allows to move among scenes.

Most of the players have the capability to move during playback. This feature consists in jumping in time forward or backward for a fixed time lapse. It allows users to go backward or forward but there is no easy way to go to specific moments of the video. Very often, users try use this feature to go to the beginning of the scene, or to skip a scene. This means that user behaviour is based on the content they are watching.

So, if the user behaviour is content-based, a nice feature to expect in a player is also content-based navigation. This capability of *smart navigation* can let user go to the beginning of a scene or directly to the next scene, independent of how many seconds it represents. By knowing the position in time of every scene, the player can provide this feature of scene navigation. It is one of the main improvements over a traditional player that the system can provides and is considered one of the main contributions of this project to the goal of obtain a *smart player*.

Shot navigation can be also useful, especially when listening dialogues and the user didn’t understand some part of the speech of an interlocutor. In dialogue scenes camera shots normally follow the intervention of each character. Moving from shot to shot in dialogues is equivalent to moving between interventions in the conversation, so it is useful to go back if viewer missed out some words.

As these features are very interesting, the metadata visualization tool should present the scene and shot information of the video as well as buttons for scene and shot navigation. However, scenes are more significant than shots to the overall video analysis, so it could be presented also in the seek bar, as marks at moments of scene changing, this is shown in Fig. 4.9.

Fig. 4.9 Scenes of video in seek bar

This provides visual information about scene duration and quantity. For doing so, a chapters text file needs to be loaded in video initialisation. The file used for this purpose is generated in metadata loading, as shown in Fig. 4.1. The structure is as simple as a common subtitle file, as it can be seen in Code 4.2.

```
1
00:00:00.01 --> 00:01:18.19
Scene 1

2
00:01:18.19 --> 00:02:33.13
Scene 2

3
00:02:33.13 --> 00:04:14.00
Scene 3
```

Code 4.2 Structure of chapters.txt file

The file of scenes, which are interpreted as chapters in the player to present the marks in seek bar of the video, consists of a consecutive numbering, followed by the start and end position of the scene and finally the name of the scene.

4.3.3 Objects and Places

As stated above, the objects and places are recognised by the processing software analysing the individual keyframes of the video. In each analysed keyframe, five objects and five places are detected, so there will always be something to show on the screen. The only changing parameter will be the level of confidence with which each of these places and objects has been found. But the actual value of this confidence is not so important, but the rank position of each.

Unlike face detection, when objects and places are detected in a keyframe the neural network does not give us any information about its exact position on the screen. This factor simplifies the work for the metadata visualization tool, because we only have to show the objects and places detected at the current position of the video, but not their location on the screen.

Therefore, the best way to show this information will be in the form of lists which will be updated automatically while the video is playing, showing all the time the objects and places that are being detected in the video. This way of presenting the information of objects and places in the video is valid based on the assumption that the temporal distance between two adjacent keyframes is small enough to consider that the objects and places of any point of the video are the same as the detected in the closest keyframe to this point.

The video player used for the implementation of the metadata visualisation tool has an event that is triggered while the player is playing. Using the activation of this event, it can be implemented a function that obtains the position of the video at that moment, search for the nearest keyframe and obtain the places and objects from it. It was already said that for continuous play it is very easy, but when jumping or navigating within scenes or shots, this function will be called and the player performance can be affected by the time consumed searching the new closest keyframe. Therefore it will be important that this function has a rapid execution. That is why in Section 4.3.1 it was shown how the search of the closest keyframe to any position of the video was optimised because it is a critical point in the execution of this function shown in Fig. 4.10.

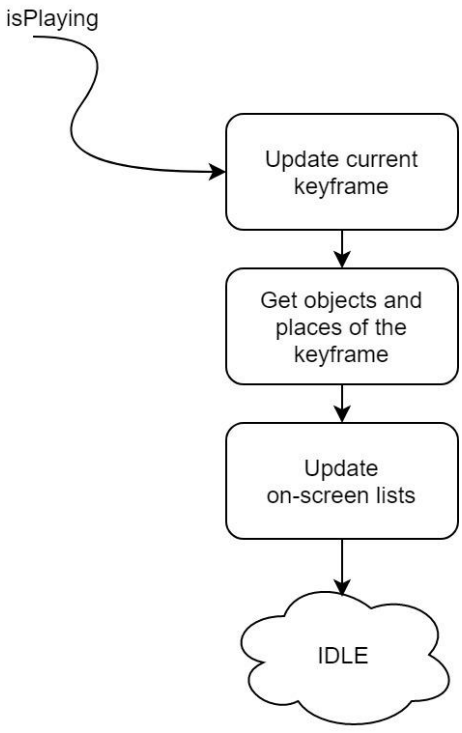


Fig. 4.10 Flow chart of objects and places real time update

If during the processing of the video many keyframes were analysed, meaning that the time distance between keyframes is too short, this information changing on screen in real time can become a bit chaotic, especially in those objects and places detected with less confidence.

4.3.4 Faces

The information offered by the processing software about faces is much more specific than the information offered about objects and places. In the case of faces the metadata specifies not only in which keyframe were found but also their position on the screen.

The main problem with the faces is that, unlike objects and places that tend to change little in the scenes and camera shots, people in video change a lot of position, either by movements of the camera angle or by the character's body language. It makes it very complicated to estimate the exact position of a face on the screen taking as reference the position of the face in the closest keyframe.

That's why it is convenient to offer the list of faces found on the screen in real time, but its position only when the video is paused. In addition, this way it would not have to deal with the unambiguous association of faces according to facial recognition when there is more than one on screen at a time, since in that case the problem of determining which face in the next frame corresponds to each face of the previous frame would have to be solved.

It is also necessary to allow the redefinition of face identifiers (*personId*) in order to use in the video identifiers that offer more comprehensible information than the listed labels that are automatically assigned by the processing software. This is perhaps the only functionality of the metadata visualization tool that modifies the original metadata, as it is considered useful to make this redefinition. The neural network assigns the identifiers according to numbers, but for a human, the change of detecting on the screen the person "4", for example, does not provide as much value as detecting on screen "Sheldon Cooper".

4.4 Drawing over video

One of the main aspects where the metadata visualization tool is highly useful is when checking the exact place on the screen where the processing software has detected the face. Because the output of the neural network are the coordinates on the face in the screen, expressed as percent of the corresponding dimension, it is very difficult to determine by a human naked eye if this position is correct. In the design stage of the metadata visualization tool one of the functionalities that was evident that was necessary to include was to show on screen the exact place where it had been recognized a face. To be able to do this in a visual way it is essential to have the ability to draw on the video.

The visual result of this implementation has already been shown in Fig. 4.5 of this chapter where it can be observed how a face detected on the screen has been shown. To make this possible it is necessary to dynamically create and destroy HTML elements on the page code.

4.4.1 Building an overlay

The first step to be able to make drawings on the image of the video through HTML elements will be to have a parent node that contains exactly the area where the image of the video is displayed. In this way it can be created HTML elements that have the visual style required for the drawing that is wanted to make and its position could be expressed in terms of to the total size of this parent node.

One of the advantages of working with JW Player is that as it is fully implemented in JavaScript it can be placed in a web page and then, during the video playback, inspect the HTML elements that make up this player. By doing so, it can be found the HTML element that is the perfect parent node for the elements that will perform the drawings on the screen.

At least in this first version of the metadata visualization tool it is only wanted to show on the screen the faces that have been detected. Therefore it will be enough to have the functionality to be able to draw a rectangle of the desired dimensions inside the screen to be able to mark in this way the position of the face. This rectangle should be transparent with edges so that it does not cover the face but only signals its position, so it can be called a *bounding box*. Besides, it will also be convenient to show along with the rectangle a short text that represents a unique identifier of the detected face. This way it also allows to test the facial recognition capability of the processing software. This functionality has been already shown in Fig. 4.5.

4.4.2 Handling dynamic bounding boxes

The main issue with dynamic creation and destruction of HTML nodes is that they must be referred by a unique identificator. There is no problem when dealing with just one face, since it is the only node, but, as shown in Fig. 4.11, there can be more than one detected face in a frame.

Fig. 4.11 Multiple faces detected in a frame

For being able to create and destroy HTML elements it is important to have an identifier assignation mechanism. A random identifier generation method was implemented. Integer numbers from 1 (one) to 1,000,000 (one million) are used, so the visualisation tool has a limit of faces draws in a frame has a maximum limit of 1,000,000 draws.

It is said “draws” and not “bounding boxes” because even when currently the player just draw bounding boxes, new features can be implemented in a future, and the identifier assignation method will be reused.

When it is needed to draw a bounding box on the screen, it is generated a random identifier for it. The program has all the time a list of the identifiers that are in use. Then, when a new bounding box is created, the program generates other random identifier and checks that is it not already in use, by comparing against the list. The odds of a collision are really low, but it is still possible, and this technique covers that possibility.

4.5 User interface

The user interface of the metadata visualization tool is the one presented in Fig. 4.12. It is graphically distributed into seven sections:

- Player options
- Video player
- Seeker
- Position information
- Object information
- Location information
- Faces information

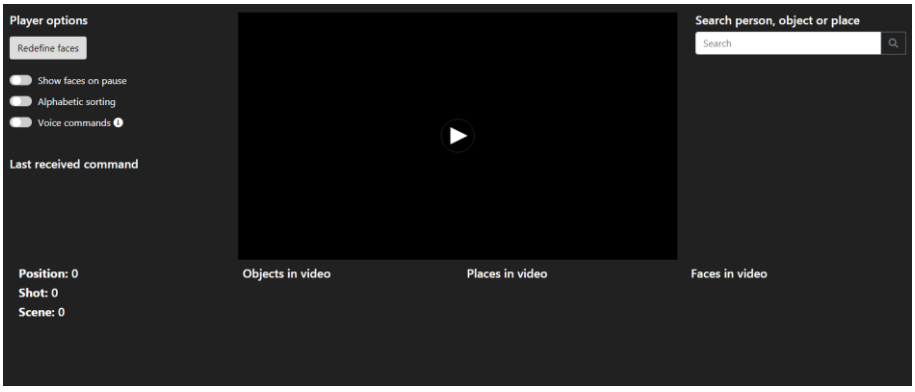


Fig. 4.12 Metadata visualisation tool user interface

The elements on the player sides are intended for user interaction. On the left, some options that can be activated or deactivated at any time. On the right, the capability to search in the video using the metadata information.

For a better understanding of these elements, they will be explained in deep separately.

4.5.1 Player options

This is the section of the user interface that allows more user interaction and controls player behaviour related to metadata. As the panel of player options has more than one functionality, a separate explanation of each of them will be provided:

Face redefinition:

When the button to redefine faces is clicked, a dialog like the one presented in Fig. 4.13 will be displayed.

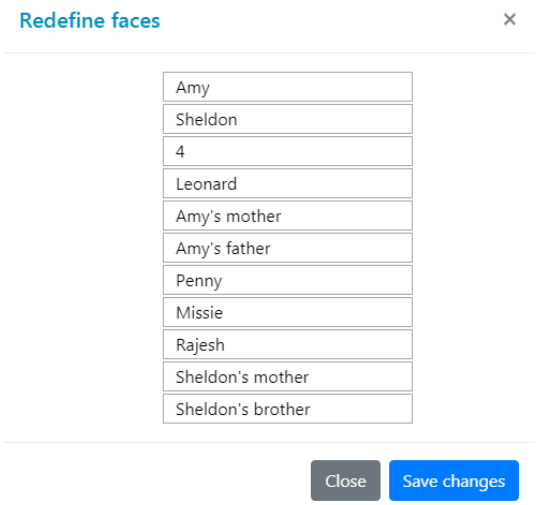


Fig. 4.13 Face redefinition dialogue box

This box presents a list of all the faces that have been detected in the video. Each of the faces shown in the table are editable, so that you can give another name to a specific face. This functionality is useful because the processing software labels the faces numerically, therefore, through this tool, it is possible to replace the numerical labels of the faces with a name that provides more information.

In addition, it would allow to unify faces. For example, if the processing software mistakenly identified the same face as two different ones, the same name could be given to both, and after saving the changes, they would be interpreted as the same face. When saving the changes in this dialog, the metadata is overwritten on the server, so the changes are persistent and it will not be necessary to redefine them each time the tool is accessed.

This phenomena of the same person with different id’s can also happen not due to an error. Very often, what actually occurs is that there is an id for the person looking at the camera, with the complete face visible, and other id for the person looking at almost ninety degrees respect the camera, with just the half of the face visible.

This feature overwrites not just the object in memory and the JSON file, but also the original XML file, and this way it allows to improve the original metadata.

Show faces on pause:

This option enables or disables the functionality of showing the faces on the screen when the video is paused. The result of enabling this function is already shown in Fig. 4.5 and Fig. 4.11. It is important to remark that faces aren't shown during playback. The performance of this functionality is very dependent on how close are the keyframes of the video, since the information relative to the position of faces is only available in keyframes.

Alphabetic sorting:

This feature controls the order in which the objects and places are shown. By default, the sorting order of these elements is the confidence level they were found with. It means that the first object or place is the one that the processing software had detected with more confidence.

But turning on the alphabetic sorting, the sorting order changes to alphabetically. This is useful because sometimes the objects and places detected on video seems a bit chaotic and it could make user thinks that the processing software is detecting objects randomly. But what actually happens is that the confidence level of the detected object and places is oscillating too much, and the sorting changes, for the same items. If this is the case, by sorting them alphabetically, the list will seems more stable.

Voice commands:

By turning on this feature, the program will start listen voice commands. This feature will be explained more deeply in Section 4.6. This option also has in information icon that by clicking it, the available voice commands are shown.

As part of the voice commands feature, at the end of the player options section in the user interface there is the **Last received command** that shows the last voice command that was received and interpreted by the player.

4.5.2 Search on video

As shown in Fig. 4.14, the search functionality work as a search engine in the metadata of the video.

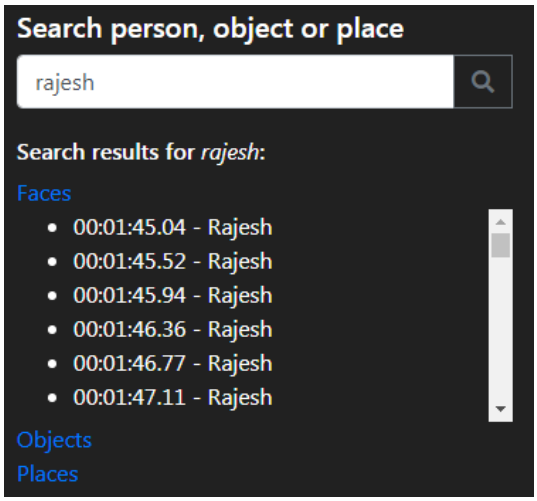


Fig. 4.14 Search in video

The user can search for faces, objects and places. The results will be offered classified by this three types. Each result is shown with the position in the video where it was founded. By clicking on a specific result line the player will navigate to that position.

The search is not case sensitive and it there is no need to put the complete word, for instance; “Rajesh” will be found when looking for “rajesh”, “raj” or “aje”.

4.5.3 Position information

This section shows the current position in milliseconds, the current shot and the current scene. It must be taken into account that the scene and shot counters starts at zero. For instance, in Fig. 4.15 it can be seen that the video is in the second 80.63, which corresponds to the second scene, in the 23rd shot.

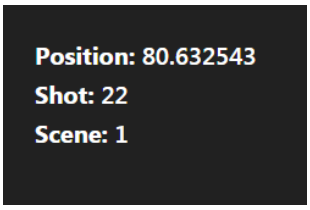


Fig. 4.15 Position information of the video

This, as all the bottom content of the user interface, is a read-only information and is helpful to check in real time the transitions between scenes and shots. The position in milliseconds isn’t rounded.

4.5.4 Objects and places

Objects and places are shown in the same way. As it can be seen in Fig. 4.16**¡Error! No se encuentra el origen de la referencia.**, the way to present this information is by using to lists.

Fig. 4.16 Objects and places information of video

In the case of this example, the sorting is by confidence level, and the information presented is coherent with the image on screen. These lists are updated by the routine illustrated in Fig. 4.16 and they are useful to check in real time the detected objects and places, not just in a single frame, but while playing. This allows to check if the detected objects and places are stable.

4.5.5 Faces in video.

Unlike objects and places, the faces list has no fixed length. In can be completely empty or contain as many faces as were detected in the current frame. The faces detected in Fig. 4.17 corresponds to the video frame presented in Fig. 4.16.

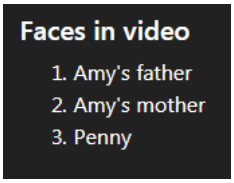


Fig. 4.17 Faces detected on a frame

This list is always alphabetically sorted for best user readability. It is also updated in real time, no matter if the functionality of showing the faces in pause is activated or not.

4.6 Voice control

One of the aspects in which this project wants to go a step further is in the control of the player. In the previous sections was already explained how to use the

information of scenes and shots to implement a content –based navigation. The other line that is going to be developed is with respect to the interface with the user. The player will allow remote control without using any device, but the voice.

4.6.1 Hardware requirements

For implementing this kind of feature a key hardware device is obviously needed: a microphone. And almost all portable devices have one, from smartphones to laptops. However, this tool has been implemented not thinking in mobile navigation, but for desktop use. So the target devices are laptops and desktop computers and since these last ones doesn't commonly have a microphone by default, it is good to specify it, because the best user experience will be in a device that has it.

The behaviour of the voice-controlled player will be strongly dependent on microphone quality and acoustics. So, good flat impulse response in the spectral region of human voice is required. Ideally, it should have perfectly flat gain from 300 Hz to 8000 Hz and be used with a sampling frequency as much greater than 16,000 Hz as possible, to be far from Nyquist limit. The specified frequency range is the recommended for speech signals [4].

Virtually any sound card that comes standard on a common computer will meet these requirements. The default sampling frequency is 44.1 kHz and the frequency response varies between manufacturers and models, but in general in the spectral region of the human voice are usually quite flat, as one of the main use cases covered by manufacturers is the video calls, where human voice has to be streamed with quality. However, the performance of this feature may vary from one device to other, due to differences in hardware.

4.6.2 Taking control of the microphone

One of the main problems that this software will face is to be able to use hardware resources of the user's computer. When specific applications of a platform are developed and a language is used that has libraries in the operating system in which it is located, there are methods to take control of the hardware resources of the device in which the program is being executed. However, in the case of JavaScript, the program is not running on the operating system's own platform, but inside a browser, therefore, as we had previously mentioned, it was not possible to access the user's file system this time it is neither able to take control (at least directly) of hardware resources of the computer that is being used.

Browsers allow some features with computer's hardware, but the situation becomes even more complicated when it is taken into account that not all browsers implement in the same way the request for access to hardware resources.

The functionality that would be needed to implement this voice control feature is to be able to obtain audio samples from the microphone of the device. These audio samples could be sent directly to a neural network that was trained to recognize samples belonging to the predefined voice commands. Of course, it

would not be deficient to continuously send all the audio samples received by the microphone of the device. In these cases, a voice detector is implemented with which it is possible to determine when a voice command is being received and only the samples belonging to the interval of time in which that command was received would be those that would be sent to the second stage of the process in which the neural network acts. This type of detectors are generally implemented through an energy detector. Its basic procedure is to filter the signal to stay only with the spectral region in which the human voice is located and an energy threshold is defined in this regions. When the received signal energy exceeds that threshold, it is estimated that a voice command is being received or at least there is a human voice in the recording. It will be the task of the neural network that comes next to determine if human voice recording corresponds to a voice command or not.

When the ways of implementing this functionality began to be evaluated, there were searched for libraries that implements at least partially the stages of audio sampling, speech detection and command recognition. From that search arose *Annyang* which is a JavaScript library that supports a very high level of abstraction using the Google speech recognition engine.

As shown in Fig. 4.18, it can be seen that using this library, the stages for obtaining samples of audio, voice detection and speech to text conversion are covered by the Google speech recognition engine.

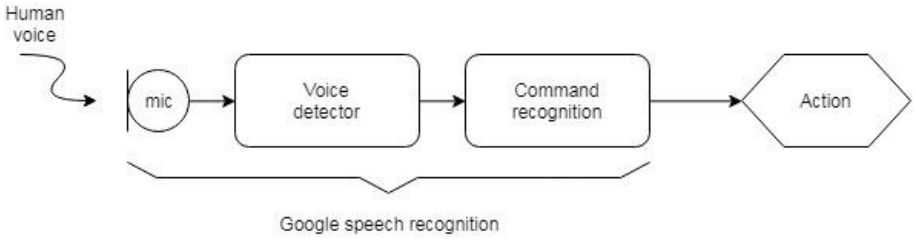


Fig. 4.18 Voice command implementation using Google speech recognition

It is expected that the performance of this implementation is not as good as performing each of the stages individually. It happens especially in the command recognition stage where if it were implemented by a well-trained neural network to recognize a finite set of commands, the result is presumably better than that of a speech-to-text engine which should be able to recognise any word of the language.

However, there are two fundamental reasons to opt for this strategy. The first is that the processing of the voice is entirely delivered to a tool developed by Google, which inspires confidence. The second element that supports this solution is that it greatly simplifies the implementation of voice control, which is always desirable. Finally the experimentation will determine if this solution is valid or not. It must be borne in mind that the use of this library implies a disadvantage which is that the system will no longer be able to work offline, at least not if it is wanted to use voice commands. Of course, the rest of the functionalities will remain functional in a disconnected environment as long as it runs on a local web server.

4.6.3 Commands optimisation

If it were an own neural network it would be very easy to know which are the aspects that would allow the optimization of its operation. A reduced commands list and commands that incorporate several words to differentiate one from the other would be the two fundamental aspects to take into account to optimize the functioning of the system facilitating the work of the neural network.

In an environment where a voice-to-text engine tool is used, the amount of commands that the system can recognize is not relevant. This is because the commands are not directly recognized from the audio samples but from the output of the speech-to-text engine stage.

By this same reasoning could be interpreted that it is not important the number of words that make up the voice command. However, using voice commands that have more words will be beneficial for the speech presence detection stage since very short words could not be detected and not even be translated into text. That is why it has been agreed to use in all commands a first word “player” that is actually unnecessary but is used to lengthen the voice command over time and facilitate its detection.

CONCLUSIONS

This project has achieved the implementation of a tool for the visualisation of video metadata. All the objectives that were drawn at the beginning of the work are considered fulfilled.

Since the product of this project is software, the environmental impact of it will be measured according to the environmental impact of the hardware used to make it work. It is the responsibility of whoever selects the web hosting where the tool will be hosted, taking into account not only the performance and economic requirements, but also the environmental commitment. It will always be wanted to work with a provider that presents guarantees of origin in the energy used for powering servers.

This tool allows the visualization of the metadata obtained from the processing of a video. One of the elements that can present an important ethical and legal nuance is the recognition of faces. The processing software generates disassociated data, where the detected faces are identified only by labels. However, through the identifier redefinition tool, you can manually associate a person's name with your face. This will provide enough information for this person to be recognized by the neural network. In addition, as it is a tool that can be published openly on the Internet, the information regarding the person and his face will be public to the whole world.

Due to what has been stated in the previous paragraph, it is evident that this software can performs a processing of personal data. This project is committed to respect the regulation (EU) 2016/679 of the European Parliament and of the Council of April 27, 2016 (GDPR) [5]. In the case that an implementation of this software will handle personal data, protected by the GDPR, the person in charge of its implementation is responsible for offering all the information to data subjects and guaranteeing compliance with the lawfulness of processing and its transparency, deciding in each case the need to appoint a data protection officer.

There are also improvements and developments of functionalities that this project proposes for future developments. In some cases, the main ways to carry them out are proposed. The main proposed developments are:

Show faces during playback.

In the same way in which the faces are shown when pausing, it would be interesting to implement the functionality of tracking the faces on the screen during playback. This functionality is highly dependent on the keyframes rate analysed. In its implementation there will be two fundamental ways, make the linear translation of the bounding box one point to the next or perform an interpolation of the position of the face in each frame from the keyframes analysed. Both ways must be tested, to define if the computational cost of the second one is justified by an improvement in the tracking of the faces.

Automatic face recognition with former faces stored.

As discussed in Section 2.3.3, a vector of numbers (*face_embedding*) corresponding to the face identification appears in the face metadata. The similarity between faces is measured through the Euclidean distance between these vectors. By renaming a face in a video, it could be implemented the ability to save the face, not only for this video, as it is currently done, but for future videos. In that way, the metadata visualisation tool itself could evaluate if one of the faces of a new processed video is sufficiently similar to a face detected in a previous video and would offer the user the possibility to rename it automatically.

Overall statistics and high level analysis.

The information collected in the metadata is sufficient to be able to make statistics about the appearance of people in the video. It could automatically define who the protagonists are, or which people make up a couple during the video. Basing the analysis on the detection of faces in frames.

You could also determine situations, for example, in the scene presented in **¡Error! No se encuentra el origen de la referencia.**, there are the faces of three people, the places detected with greater confidence correspond to interiors of vehicles and between the objects are "seat belt" and others types of cars that are seen through the car windows. With this information, it could be concluded that these three people are making a car trip.

Gesture control.

In the same way that the current player implements voice commands, you could experience control through gestures. Laptops have both microphones and cameras, so the hardware requirement will not be a problem. In addition, in the same way that a browser can request access to the microphone, you can request it to access the camera. In this case it will be useful to explore if there are currently open source solutions for its implementation in JavaScript.

ACRONYMS

AI	Artificial Intelligence
CSV	Comma Separated Values
DL	Deep Learning
GDPR	General Data Protection Regulation
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
ML	Machine Learning
NPM	JavaScript package manager (Node Package Manager)
PHP	HyperText Preprocessor
XML	Extensible Markup Language

REFERENCES

[1] Instituto Nacional de Estadística, “Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación en los Hogares.” 07-Nov-2018.

[2] Instituto Nacional de Estadística, “Encuesta Continua de Hogares.” 12-Apr-2018.

[3] W3C, “Extensible Markup Language (XML) 1.0 (Fifth Edition),” 07-Feb-2013. [Online]. Available: https://www.w3.org/TR/REC-xml/. [Accessed: 23-Dec-2018].

[4] A. Sehgal and N. Kehtarnavaz, “A Convolutional Neural Network Smartphone App for Real-Time Voice Activity Detection,” *IEEE Access*, vol. 6, pp. 9017–9026, 2018.

[5] “REGULATION (EU) 2016/ 679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL - of 27 April 2016 - on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/ 46/ EC (General Data Protection Regulation),” p. 88.

[6] J. Dalton and A. Deshmane, “Artificial neural networks,” *IEEE Potentials*, vol. 10, no. 2, pp. 33–36, Apr. 1991.

[7] M.-C. Su and M.-T. Chung, “Voice-controlled human-computer interface for the disabled,” *Comput. Control Eng. J.*, vol. 12, no. 5, pp. 225–230, Oct. 2001.

[8] “v8 Javascript API Reference - JW Player JavaScript API Reference.” [Online]. Available: https://developer.jwplayer.com/jw-player/docs/javascript-api-reference/. [Accessed: 28-Jan-2019].

[9] I. Aljarrah and D. Mohammad, “Video content analysis using convolutional neural networks,” in *2018 9th International Conference on Information and Communication Systems (ICICS)*, 2018, pp. 122–126.

[10] “Cinemaesencia: Escena.” [Online]. Available: http://elokuvantaju.uiah.fi/spanish/study_material/screenplay/kohtaus.jsp. [Accessed: 01-Jan-2019].

[11] C. Easttom, *Advanced Javascript*. Jones & Bartlett Learning, 2010.

[12] D. Flanagan, *JavaScript: la guía definitiva*. Anaya Multimedia, 2007.

[13] *Understanding Advanced JavaScript*. Smashing Magazine, 2013.